# TASK DRIVEN AUTONOMOUS ROBOT NAVIGATION WITH IMITATION LEARNING AND SENSOR FUSION

A THESIS

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENT

FOR THE DEGREE

MASTER OF SCIENCE

BY

JUNHONG XU

DR. SHAOEN WU- ADVISOR

BALL STATE UNIVERISTY

MUNCIE, INDIANA

MAY 2018

# ABSTRACT

**THESIS PAPER:** Task-Driven Autonomous Robot Navigation with Imitation Learning and Sensor Fusion

**STUDENT:** Junhong Xu

**DEGREE:** Master of Science

**COLLEGE:** Sciences and Humanities

**DATE:** May 2018

**PAGES:** 48

The ability of interacting with dynamic and complex environments with minimal prior knowledge is a key challenge in mobile robots. The interaction can be in the form of avoiding dynamic obstacles or following human instructions. Such robotic system have various applications such as search-and-rescuer, autonomous delivery, or self-driving. Designing and implementing controllers for such robotic system requires tremendous efforts and always prone to error. Rather than programming such controller, it will be more beneficial to allow the robot to learn from others' and its own experiences.

In this thesis, we focus on enabling the mobile robot to perform different tasks based on visual inputs in indoor environments via imitation learning. Imitation learning is a data-driven approach that uses expert demonstrations to train a policy that performs the demonstrated task. However, it requires heavy supervision from human experts. In addition, it is hard to perform multiple task using the same model. Our first framework focuses on reducing human supervision. It is an extension of Dataset Aggregation (DAgger) method, in which we use the sensor fusion technique to allow the robot to learn a navigation policy in a self-supervised manner thus minimizes human supervision.

The second framework learns a multi-task policy using shared information between the related tasks. It performs different tasks based on human instructions. These tasks are navigating to different indoor environments or exploring the current one. We performed an extensive collection of experiments for each framework and demonstrates that the proposed frameworks are able to achieve high performance and even surpasses human demonstrator in some scenarios.

# Contents

**1 INTRODUCTION** **1**

**2 RELATED WORK** **4**

2.1 Deep Learning . . . . . 4

2.2 Learning from Demonstrations . . . . . 4

2.3 Deep Reinforcement Learning . . . . . 6

**3 Multi-Sensory Semi-Supervised Imitation Learning** **7**

3.1 Robotic Platform . . . . . 7

3.2 *MS3L* Policies . . . . . 9

3.2.1 Human Policy . . . . . 9

3.2.2 Sensor Policy . . . . . 9

3.2.3 Navigation Policy . . . . . 10

3.2.4 Recording Policy . . . . . 12

3.3 Training Protocol . . . . . 14

**4 Shared Multi-task Imitation Learning** **15**

4.1 Problem Formulation . . . . . 15

4.1.1 Traditional Imitation Learning . . . . . 16

4.1.2 Multi-Task Imitation Learning . . . . . 16

4.2 Network Architecture . . . . . 17

4.2.1 Image Feature Extractor . . . . . 17

4.2.2 Shared Multi-headed Policy . . . . . 18

4.3 Training Procedure . . . . . 20

4.3.1 Dropout and Data Augmentation . . . . . 20

4.3.2 Noise Injection . . . . . 21

**5 EXPERIMENTS** **22**

5.1 MS3L Performance Evaluation . . . . . 22

5.1.1 Training Evaluation . . . . . 24

5.1.2 Recording Threshold $\beta$ . . . . . 26

5.1.3 Performance of Navigation After Training: Comparison with Baselines . . . . . 27

5.2 *SMIL* Performance Evaluation . . . . . 30

5.2.1 Task Description . . . . . 30

5.2.2 Experiment Setup . . . . . 30
5.2.3 Results . . . . . 32

**6 CONCLUSION** **36**

**REFERENCES** **38**

# 1 INTRODUCTION

One of the main challenges in robotics is how to develop robots that are able to interact within a dynamically changing environment and perform different tasks with minimal prior knowledge . It requires the robot to percept the environment and make decisions accordingly. Traditional methods rely on accurate manual modeling for each task. For example, in the task of indoor navigation, one of the most widely adopted approaches is SLAM, which is a two-stage approach consisting of perception and action stage [37]. A global map is built using on-board sensors in the perception stage. This map is then given to a motion planner to make predictions [25, 36]. This approach is environment-dependent and requires tremendous efforts to establish the environment maps. It also needs intensive memory to store the map information.

In contrast, learning-based methods simplifies the need of manual modeling. They directly learn a policy that maps a sensor input to the corresponding control command. Because of the advances of deep learning models, especially convolutional neural networks (CNNs), learning control commands directly from raw images becomes possible [17]. Deep reinforcement learning (DRL) [22] is one such method that combines deep learning with reinforcement learning. Although it has yielded many successful outcomes [19–21, 31], it requires robots to learn from trial-and-error that is impractical and too expensive in real world tasks. Therefore, many works only employ DRL in simulated settings [12] to avoid hardware damaging and long time training. The learned policy is then transferred to real world scenarios [30, 35]. However, the discrepancy between simulated environments and the real world is large and still requires fine-tuning. Meanwhile it is very likely to cause damage to robots.

End-to-end imitation learning methods are another branch of learning-based control and it has been employed to resolve complex robotic tasks such as manipulating objects [24], navigating to a target position [23], and self-driving vehicles [3]. Imitation learning converts sequential decision tasks to supervised learning problems, where a policy is trained to minimize the errors between the predicted actions and the ones taken by an expert demonstrator. It is more simple and data efficient than reinforcement learning thus incur less damage while applying to real robots. However, there are three major challenges in imitation learning: **(1)** data distribution mismatch between states encountered by a trained policy and an expert policy, **(2)** difficulties of querying an expert policy in real-world scenarios, and **(3)** only learns single task per model. This thesis proposes two frameworks that addresses these challenges.

The first two problems are usually solved by DAgger algorithm [26] , which iteratively collects training examples using both an expert and a trained policy. This approach however requires a human supervisor to provide correct control commands given an observation encountered by a trained policy, which is expensive and inaccurate [27]. We propose a framework, ***M**ulti-**S**ensory **S**emi-**S**upervised **L**earning (MS3L)*, which is based on imitation learning augmented with sensor fusion to overcome the challenges of applying DAgger to real robots. It has two key innovations:

- One is to employ various types of sensors, including both imaging and non-imaging, in a deep learning framework to minimize human involvement in that it only requires ONE iteration of human supervision to initialize a navigation policy.

- The other is that, after initializing the navigation policy, *MS3L* uses a suboptimal *sensor policy* to label the observations encountered by a mobile robot at its own,

which completely eliminates the need of querying an expert policy in literature solutions. To reduce the adversarial effect of learning from the suboptimal policies, we design a *recording policy* based on the safety policy [40], which controls the degree of information learned by the navigation policy.

Imitation learning traditionally only learns one task per model. This restricts robots to execute complex actions. Our second framework is called ***S**hared **M**ulti-task **I**mitation **L**earning (SMIL)*, which gives robots the ability to perform different tasks while navigating in indoor environments.. The framework is based on multi-task learning (MTL) [4] and aims at solving the last problem mentioned above. Although MTL has been researched for a long time, mutli-task imitation learning has rarely been researched until recently. The proposed framework uses a shared CNN to learn a environment model that extracts image features. Different subtask policies are represented by a multi-headed fully connected network which inputs are from the last layer of the shared CNN. While previous work [5] does not consider the relevance between each subtask, the proposed framework makes use of the relevant information across sub-tasks. During training, instead of using on-policy learning as presented in *MS3L*, we apply off-policy imitation learning, where robots learn from an noise injected expert policy [15]. During inference, the framework switches between sub-policies based on human commands.

In the rest of this thesis, Section 2 reviews the related work and background of the topic. Then, Section 3 discusses the detailed design and training pipeline of *MS3L*. *SMIL* is introduced in Section 4 including network architecture and detailed training procedure. Next, Section 5 presents extensive performance evaluations of *MS3L* and *SMIL* in real indoor environments. The paper is finally concluded by Section 6.

# 2 RELATED WORK

## 2.1 Deep Learning

Deep learning is a broad set of learning algorithms that usually consists of different types of neural network layers such as CNNs used for computer vision or long short-term memory (LSTM) used for sequential learning [10]. With many of these layers stacking together, deep learning algorithms can become powerful function approximators that is able to learn representation from input data. They map a fixed size input such as images to a fixed size output such as classification probability or real-valued numbers. The mapping is transformed by a set of stacked layers. Each of these layers learns different representations of input data [17].

With the ability of learning representations, deep learning has made much progress in computer vision. Many network architectures have been proposed to further increase the performance of deep learning models [9, 32, 34]. In our work, we utilize the power of these deep models to map raw images to robot control commands. However, the success of training these deep models are based on large amount of data like ImageNet [6, 29]. In contrast, there is no universal dataset for robotics because it is hard to collect and label the data needed for training, which can be problematic while training a robotic task.

## 2.2 Learning from Demonstrations

One of widely used imitation learning algorithms in the literature is unanimously based on a pioneer DAgger algorithm proposed by Ross *et al.* [26] to iteratively train a policy that imitates a certain expert policy. It is an example of on-policy learning algorithm, where

the robot learns from its own trajectories. This algorithm has been used in many robotic problems [7, 14, 27]. It requires some expert policies to be queried, which is impractical and too expensive in real environments especially when human supervision is required. Ross *et al.* use DAgger to train a drone to fly in forests and avoid collisions [27]. The human supervisors are provided with partial feedback to correct actions of trained policy off-policy. While this solution aims at reducing supervisor's burden, it still does not solve the problem of querying a human operator. Laskey *et al* proposes an approach to use a hierarchy of supervisors to learn grasping policy [14], which actually requires more on the burden of human supervisor. There are many works extending DAgger algorithm and focusing on the improvement of query efficiency to an expert policy [16, 40]. These works are the most relevant to ours in a way that they constrain training data with some query metrics. These methods however assume that an oracle is always available and easy to be queried, which is often unlikely in real environments. In addition, the issue of noisy sensor measurements is not addressed in these works, which definitely degrade the performance of a trained policy.

Another method of imitation learning is based on off-policy training, where the robot learns from trajectories generated by other policies. DART [15] is an off-policy learning algorithm that injects an optimized Gaussian noise into supervisor's actions to simulate errors caused by robot's policy. The supervisor is forced to give the correct control commands to avoid failures. The robot then learns to recover from undesired states from these corrective commands.

In this work, we use both of the above techniques to collect data and train our model. They will be presented in Section 3 and 4.

The above mentioned algorithms only consider completing one task at a time, but this is not enough for many robotic tasks. There have been a few works that address multi-task imitation learning. Hausman et. al. proposed a multi-model imitation learning framework that learns to separate video segments into different skill trajectories and imitate jointly [8]. In [5], the authors proposed a similar framework to ours that learns sub-policies using multi-headed network in autonomous driving setting. However, they did not consider the relevant information among tasks. Our proposed framework learns relationships across tasks by combining learned features across sub-policies. By learning these relationships, the model is able to learn a more general representation of the navigation task [28].

## 2.3 Deep Reinforcement Learning

Recently, deep reinforcement learning (DRL) [22] has been attracted much attention in the robotic control field. Many works based on DRL have been performed to address robotic navigation tasks. In [12], the authors train a DQN (deep q learning) agent [22] to cross an intersection in simulation. Another work proposes a simulated environment to train a DRL agent to reach a target position in indoor environments [41]. Lillicrap et al. [19] proposes deep deterministic policy gradient (DDPG) to train an agent to avoid dynamic obstacles in simulation. These methods however are all constrained in simulated settings where damage to the agents is not a concern. Applying DRL to real environments is still challenging. Many researchers attempt to address this problem by transferring learned DRL policies from simulation to real world navigation tasks [30, 38]. The difference between simulated environments and the real world settings makes this

adaption difficult.

# 3 MULTI-SENSORY SEMI-SUPERVISED IMITATION LEARNING

We consider a mobile robot navigating in indoor environments of pedestrians and obstacles. The robot's goal is to navigate rapidly and safely in indoor environments. Our goal is to minimize human supervision and allow the robot to learn from its own experience. We do not assume the robot has access to a multi-stage motion planer as a reference policy as in [23].

We propose and implement a system, ***M**ulti-**S**ensory **S**emi-**S**upervised **L**earning (MS3L)*, to enable the autonomous robotic navigation. Our system combines four policies $\pi_h$, $\pi_s$, $\pi_{\theta_r}$, and $\pi_{\theta_n}$, which respectively represent human policy, sensor policy, recording policy, and navigation policy. By initializing $\pi_{\theta_n}$ with $\pi_h$ and constraining learning from a suboptimal policy $\pi_s$ with a recording policy $\pi_{\theta_r}$, the robot is able to surpass the suboptimal policy and achieve near human performance in a self-supervised manner.

In this section, we present the detail design and framework of *MS3L*, including the robotic platform, the policies, and the training protocol.

## 3.1 Robotic Platform

We have built our own robot based on commodity supplies as shown in Fig. 2. The base of the robot is an iRobot Create2[1], which has a linear velocity of [-0.5$m/s$, 0.5$m/s$] and

[1]http://www.irobot.com/About-iRobot/STEM/Create-2.aspx

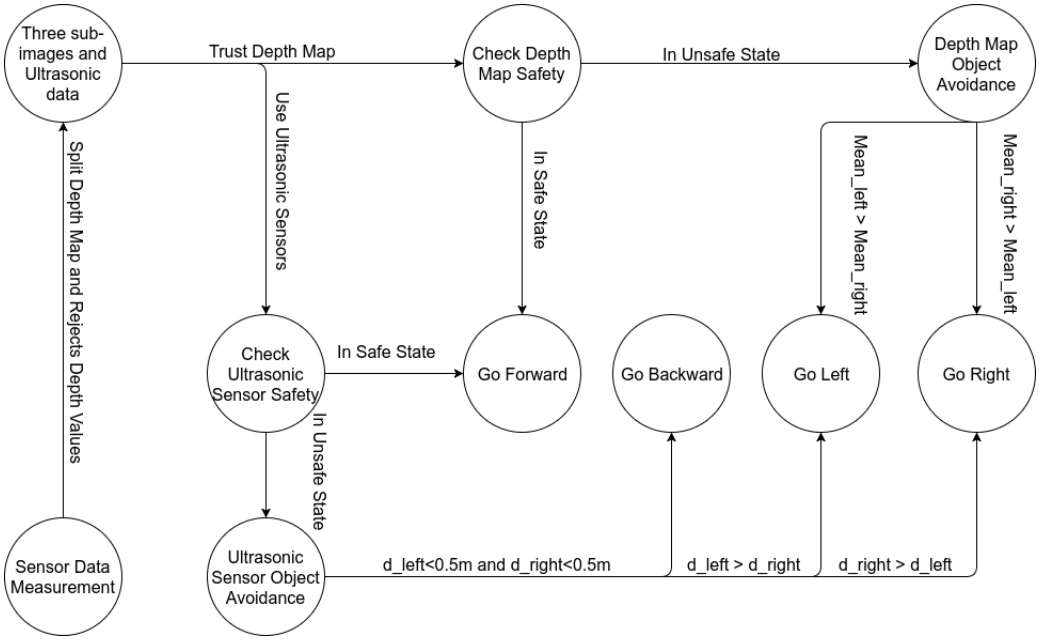



Figure 1: Finite state machine for sensor policy.

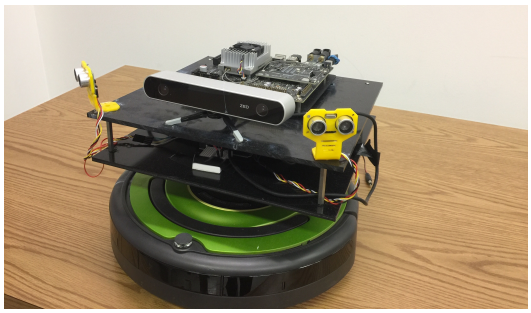


Figure 2: iRobot Create2 equipped with a stereo camera, two ultrasonic sensors, and a Jetson TX1.

an angular velocity of [-4.5$rad/s$, 4.5$rad/s$]. The robot is equipped with two ultrasonic sensors for distance measurements and a ZED™ Stereo Camera[2] to capture RGB images and as well as depth images. The ultrasonic sensors can detect objects within a distance of [5$cm$, 400$cm$] (or [2$in$, 156$in$]). The valid depth estimation of the stereo camera falls into [0.5$m$, 20$m$]. These two types of depth/distance sensors, namely ultrasonic and stereo camera, complement with each other to derive our sensor policy. In addition, a NVIDIA® Jetson TX1[3] is used as the robot's brain to make inferences, which has 256

[2]https://www.stereolabs.com/
[3]http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html

CUDA cores with 2GB memory to support a moderate-sized neural network in real-time computation.

## 3.2 *MS3L* Policies

The *MS3L* solution has four policies including *human policy* ($\pi_h$), *sensor policy* ($\pi_s$), *navigation policy* ($\pi_{\theta_n}$) and *recording policy* ($\pi_{\theta_r}$), designed for the multi-sensory imitation learning. Among those, *human policy* and *sensor policy* are also referred as *reference* policies ($\pi_{ref}$) to guide the navigation.

### 3.2.1 Human Policy

Human policy $\pi_h$ refers to a human operation that guides the robot to navigate through environments and avoid collision. The operator provides initial demonstrations using a joystick to tele-operate the robot remotely. Note that the human policy is used only ONCE in data collection for each of the navigation and recording policy.

### 3.2.2 Sensor Policy

Sensor policy $\pi_s$ is used in the self-supervised learning phase to generate suboptimal action labels. This policy generates a set of basic robot control commands: turning, acceleration, and deceleration. Unlike in simulated or controlled environments where dynamics are always known, it is not possible to obtain an accurate model of an unstructured real indoor environment. Our goal is not to use an expensive accurate range sensor like a laser device used in [23] to create an optimal depth control policy, but to build a suboptimal sensor policy $\pi_s$ upon the coarse measurements of those unreliable cheap commodity sensors.

Sensor policy $\pi_s$ is high bias and low variance due to the hand engineered control and noisy measurements in sensor data. It is regulated through a finite state machine as shown in **Fig. 1**. It takes the current depth map $M$ and ultrasonic sensor measurements $d_{left}$ and $d_{right}$ as inputs and generates a control command $a = \pi_s(M, d_{left}, d_{right})$. At the first stage, it splits the depth map $M$ into three sub-image. Then, it rejects depth values in each sub-image which has the value larger than two standard deviation of the entire depth image This checks whether we can trust the current depth estimation. If the number of rejected data is large, the robot distrusts the depth map, and rather uses the two ultrasonic sensor measurements ($d_1$, $d_2$) to calculate robot control actions. This usually happens when the robot is facing towards a plain wall or objects only appear in one camera. If the depth image is used, mean depth value of the middle sub-image is used, e.g. the value is smaller than 0.8m in our experiments, to determine whether the robot needs to take object avoidance actions. If the ultrasonic sensor is used, $d_{left}$ and $d_{right}$ are used to check whether the robot is in the safety position. Go back action is only taken when ultrasonic sensor is used because ZED stereo camera is not valid in detecting distance within 0.5 meters.

It should be noted that, $\pi_s$ is not robust and we expect the recording policy described later to constrain the robot to only learn a subset of data generated by $\pi_s$

### 3.2.3 Navigation Policy

Navigation policy $\pi_{\theta_n}$ outputs an action to avoid obstacles based only on the current RGB image observed from ZED stereo camera. To handle high dimensional observations i.e. images in our case, we parameterize $\theta_n$ as a 5-layer convolutional neural network (CNN)

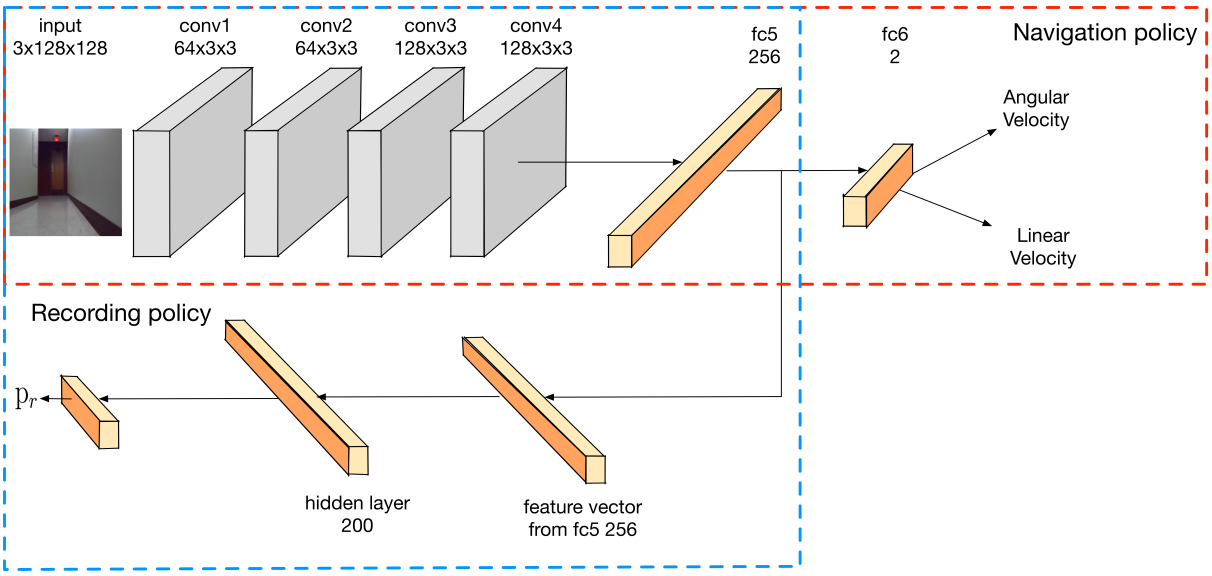



Figure 3: Our model consists of two policies: navigation policy $\pi_{\theta_n}$ and recording policies $\pi_{\theta_r}$

as shown in Fig.3. We adopt VGG-like architecture [32] in our design. All convolution layers have a $3 \times 3$ kernel size. The first two convolution layers have 64 channels followed by a max pooling layer. The last two convolution layers have 128 channels followed by a fully-connected layer with 256 neurons. ReLU activation functions are applied to all layers. We normalize iRobot Create2 linear and angular velocities between $[-1, 1]$ when collecting dataset. To bound with the this normalized scale, we add a *tanh* activation function before the output. The policy outputs 2-dimensional actions in forms of linear velocity and angular velocity. Input images are resized to $128 \times 128$ for real-time inference.

We define the navigation policy dataset as $D_{\pi_{\theta_n}} = \{(x_1, \pi_{ref}(x_1)), ..., (x_i, \pi_{ref}(x_i)), ..., (x_t, \pi_{ref}(x_t))\}$, where $x_i$ represents the observation at timestep $t$, and $\pi_{ref}(x_i)$ refers to the output of reference policies consisting of human policy $\pi_h$ that is used only in the first iteration and sensor policy $\pi_s$ that is used in the rest of training iterations. The imitation objective is

defined as

$$C_i(D_{\pi_{\theta_n}}, \pi_{\theta_n}) = 1/N \times \sum_{i=1}^{i=N} ||\pi_{\theta_n}(x_i) - \pi_{ref}(x_i)||_2^2, \quad (1)$$

where $N$ is the mini batch size for training. With this objective function, the navigation policy is trained to imitate from both human and sensor policies: $\pi_h$ and $\pi_s$.

### 3.2.4 Recording Policy

Recording policy $\pi_{\theta_r}$ is crucial to our learning framework, where $\theta_r$ is parameterized by a 2-layer fully-connected neural network. It takes a feature vector from the last fully-connected layer $fc5 = \pi^5_{\theta_n}(x)$ of the navigation policy as input and generates the probability $p_r = \pi_{\theta_r}(fc5)$ that the current observation is needed for the navigation policy to learn, as shown in Fig.3, where $x$ is observation and $\theta^5_n$ represents the parameters of last fully-connected layer of the navigation policy. This design choice of using shared convolutional layers for navigation and recording policies is based on two factors. First, the limited GPU memory on Jetson TX1 is not powerful enough for two CNNs: one for the navigation policy and the other for recording policy. Second, the feature extracted by the CNN at layer $fc_5$ can be shared and reused by both the navigation policy and the recording policy to improve its learning speed [17].

We use the sensor policy $\pi_s$ as a suboptimal policy to self label data after the human operated pre-training stage. In order to throttle the propagation of learning errors from $\pi_s$, the recording policy only keeps those labelled data from $\pi_s$ if $\pi_{\theta_n}$ deviates far from both reference policies. We define the deviation as the squared difference between the

output of the navigation policy $\pi_{\theta_n}$ and that of reference policies ($\pi_h$ and $\pi_s$) as

$$\begin{aligned} e(x, \pi_{\theta_n}, \pi_h, \pi_s) = \gamma ||\pi_{\theta_n}(x) - \pi_h(x)||_2^2 \\ +(1-\gamma)||\pi_{\theta_n}(x) - \pi_s(x)||_2^2, \end{aligned} \tag{2}$$

where $\gamma$ is a constant between 0 and 1 that weights the relative importance of the two reference policies, e.g. if $\gamma$ is closer to 1, the sensor policy is less accounted for the deviation. With this deviation metric, a binary function indicating whether to record is defined as

$$\epsilon(x, \pi_{\theta_n}) = \begin{cases} 1, & \text{if } e(x, \pi_{\theta_n}, \pi_h, \pi_s) > \tau \\ 0, & \text{otherwise} \end{cases}, \tag{3}$$

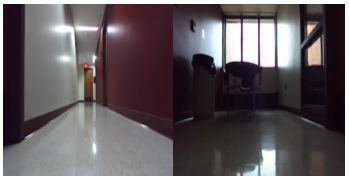
where $\tau$ is a scalar indicating the degree of error tolerated by the recording policy.

$\pi_{\theta_r}$ is trained on a recording dataset $D_{\pi_{\theta_r}} = \{\pi_{\theta_n}^5(x_1), ..., \pi_{\theta_n}^5(x_n)\}$, which is collected using extra human demonstrations, from the last fully-connected layer $fc5$. The objective function to be minimized by $\pi_{\theta_r}$ is a binary cross-entropy loss function defined as

$$\begin{aligned} C_r(D_{\pi_{\theta_r}}, \epsilon, \pi_{\theta_r}\pi_{\theta_n}, \pi_{ref}) = -\frac{1}{N} \times \\ \sum_{n=1}^{N} \epsilon(x_n, \pi_{\theta_n}) log(\pi_{\theta_r}(x_n)) + (1 - \epsilon(x_n, \pi_{\theta_n})) log(1 - \pi_{\theta_r}(x_n)), \end{aligned}$$

where $\pi_{ref}$ is the reference policies consisting of $\pi_h$ and $\pi_s$.

(a) $\beta = 0.99$



(b) $\beta = 0.5$

Figure 4: Images are uniformly sampled from datasets that have different $\beta$ values.

After the pre-training stage, observations are recorded only if $\pi_{\theta_r}(x) > \beta$, where $\beta$ is a recording threshold controlling the degree to which the robot trusts $\pi_{\theta_r}$. We refer the observations recorded by $\pi_{\theta_r}$ as hard samples. Some examples with different thresholds are shown in Fig.4. Fig.4a shows the images recorded in dangerous states i.e. turning or being close to walls. In contrast, Fig. 4b shows the images of going forward, which are not necessary for the navigation policy to learn. A larger $\beta$ places more constraints to the number of observations recorded and labeled by $\pi_s$.

## 3.3 Training Protocol

**Algorithm 1** Multi-Sensory Self-Supervised Learning

**procedure** *MS3L* Training Protocol

Randomly initialize $\pi_{\theta_r}$ and $\pi_{\theta_n}$.

**Pre-training**

Collect $D_{\pi_{\theta_r}}$ and $D^0_{\pi_{\theta_n}}$ using $\pi_h$

$\pi_{\theta^0_n} = arg\,min_{\theta_n} C_i(D^0_{\pi_{\theta_n}}, \pi_{\theta_n})$

$\pi_{\theta^0_r} = arg\,min_{\theta_r} C_r(D_{\pi_{\theta_r}} \cup D^0_{\pi_{\theta_n}}, \epsilon, \pi_{\theta_r}, \pi_{\theta^0_n}, \pi_{ref})$

**Self-supervised learning**

**for** $i = 1$ **to** $k$ **do**

Execute $\pi_{\theta^{i-1}_n}$, collect and label observations into $D$ using $\pi_s$ and $\pi_{\theta^{i-1}_r}$.

Only keep $(x, \pi_{ref}(x_i))$ pair from $D$ if $\pi_{\theta^i_r}(\pi^5_{\theta^{i-1}_n}(x)) > \beta$

$D^i_{\pi_{\theta_n}} = D \cup D^{i-1}_{\pi_{\theta_n}}$

$\pi_{\theta^i_n} = arg\,min_{\theta_n} C_i(D^{i-1}_{\pi_{\theta_n}}, \pi_{\theta^{i-1}_n})$

$\pi_{\theta^i_r} = arg\,min_{\theta_r}$
$C_r(D_{\pi_{\theta_r}} \cup D^i_{\pi_{\theta_p}}, \epsilon, \pi_{\theta^{i-1}_r}, \pi_{\theta^{i-1}_n}, \pi_{ref})$

**return** $\pi_{\theta^4_n}$ and $\pi_{\theta^4_r}$

With these policies, the training procedure of *MS3L* is performed as: the navigation policy $\pi_{\theta_n}$ is initialized by human policy $\pi_h$ and iteratively trained by a suboptimal internal policy $\pi_s$ constrained on a recording policy $\pi_{\theta_r}$. Our training procedure consists of two stages with five iterations in total, which is shown in Algorithm 1. The first stage

is a pre-training stage, where a human operator controls the robot using a PlayStation wireless controller to navigate through the environments and collect the initial recording and navigation policies datasets: $D_{\pi_{\theta_r}}$ and $D^0_{\pi_{\theta_n}}$ to initialize the navigation policy: $\pi_{\theta_r}$ and $\pi_{\theta_n}$. These two datasets are collected in opposite directions shown in Fig.8. to ensure that the recording policy correctly classifies hard observations. While the navigation policy is trained to have a basic understanding of the environment, it does not learn any complex actions such as turning or decelerating. A self-supervised learning stage is proposed to further improve the policy without human operators. At this stage, $\pi_s$ is used to generate labels for the collected data. In addition, $\pi_{\theta_r}$ is used to constrain the data collection to remove largely deviated data. Specifically, an observation $x$ and its corresponding label $\pi_s(M, d_1, d_2)$ are only recorded when $\pi_{\theta_r}(\pi^5_{\theta_n}(x)) > \beta$. This ensures that the deficient labeling resulted from noisy measurements of $\pi_s$ is minimized. Then, the navigation policy is trained upon the aggregation of the initial dataset and the selected observations. The trained navigation policy and the aggregation of recording and navigation datasets are used to update the recording policy.

# 4 SHARED MULTI-TASK IMITATION LEARNING

We first formally define the problem of imitation learning and multi-task imitation learning. Next, we present our network architecture. Finally, we present our training procedure in detail.

## 4.1 Problem Formulation

To formally define the problem, let $\mathcal{X}$ and $\mathcal{Y}$ denote recorded observations and the corresponding expert control commands respectively. To simplify the notation, we assume the environment is Markovian, meaning the current observation includes all history information of the environment.

### 4.1.1 Traditional Imitation Learning

In the traditional imitation learning setting, the demonstrations only consist of a single task. Thus, $\mathcal{X} = \{\mathcal{X}^1, \mathcal{X}^2, ..., \mathcal{X}^n\}$ and $\mathcal{Y} = \{\mathcal{Y}^1, \mathcal{Y}^2, ..., \mathcal{Y}^n\}$ represent $n$ demonstrations. It aims to learn a policy that maps an observation to a probability distribution of control command $\pi : \mathcal{X} \rightarrow \pi(\mathcal{Y}; \theta)$, where $\theta$ denotes the parameters of a weight vector, e.g. neural network. The parameters $\theta$ can be found by solving maximum-likelihood (ML): $\theta^* = \underset{\theta}{arg\,max} \sum_{n=1}^{N} \pi(\mathcal{Y}^n | \mathcal{X}^n; \theta)$. If the policy is Gaussian and it is parameterized by a weight vector $\theta$, we can rewrite ML objective into $l2$ regression problem: $\theta^* = \underset{\theta}{arg\,min} \frac{1}{N} \sum_{n=1}^{N} ||\pi(\mathcal{X}^n; \theta) - \mathcal{Y}^n||_2^2$. In all the experiment, we assume our policy is Gaussian thus we use $l2$ objective to optimize the parameters $\theta$.

### 4.1.2 Multi-Task Imitation Learning

In the multi-task imitation learning setting, observations $\mathcal{X}$ and the corresponding control commands $\mathcal{Y}$ no longer only consist of a single task demonstration. Instead, they consist of multiple demonstrated tasks and are denoted as $\mathcal{X} = \{X_1, X_2, ...X_i\}$ and $\mathcal{Y} = \{Y_1, Y_2, ..., Y_i\}$, where $X_i = \{X_i^1, X_i^2, ..., X_i^n\}$ and $Y_i = \{Y_i^1, Y_i^2, ..., Y_i^n\}$ represent the observations and control commands of task $i$. Although here we use the same num-

ber of demonstrations across tasks for simplifying notations, it needs not to be the same. In addition, we introduce the concept of task embedding denoted as $T = \{T_1, T_2, ...T_i\}$. Similar to word embedding [18], the task embedding is a feature vector for each task that embeds the high-level meaning into low-dimensional space.Therefore, the policy maps observations and task embedding to a distribution of control command and is denoted as $\pi : \{\mathcal{X}, T\} \rightarrow \pi(\mathcal{Y}; \theta)$. Instead of minimizing the objective for one task, multi-task imitation learning aims to find a set of parameters that is optimized across multiple tasks:

$$\theta^* = \underset{\theta}{arg\,min} \frac{1}{I} \frac{1}{N} \sum_{i=1}^{I} \sum_{n=1}^{N} ||\pi(\mathcal{X}_i^n, T_i; \theta) - \mathcal{Y}_i^n||_2^2, \tag{4}$$

where $I$ is the number of tasks.

## 4.2 Network Architecture

In this section we present a ***S**hared **M**ulti-task **I**mitation **L**earning* (SMIL) framework that learns to perform four tasks based on human command. As shown in Fig. 5, it consists of two modules: image feature extractor and shared multi-headed policy which are presented in the following sections.

### 4.2.1 Image Feature Extractor

We fine-tune the pre-trained ResNet-18 as the image feature extractor by excluding the classifier layer in the original ResNet-18, but preserve the average pooling layer. It is used to project raw image inputs to a low-dimensional feature space and is denoted as $f = I(x, \theta_I)$, where $f \in \mathbb{R}^{1\times1\times512}$ is the extracted feature vector, $x$ is the input image, and $\theta_I$ is the parameter set of ResNet-18. The idea of using pre-trained model has been

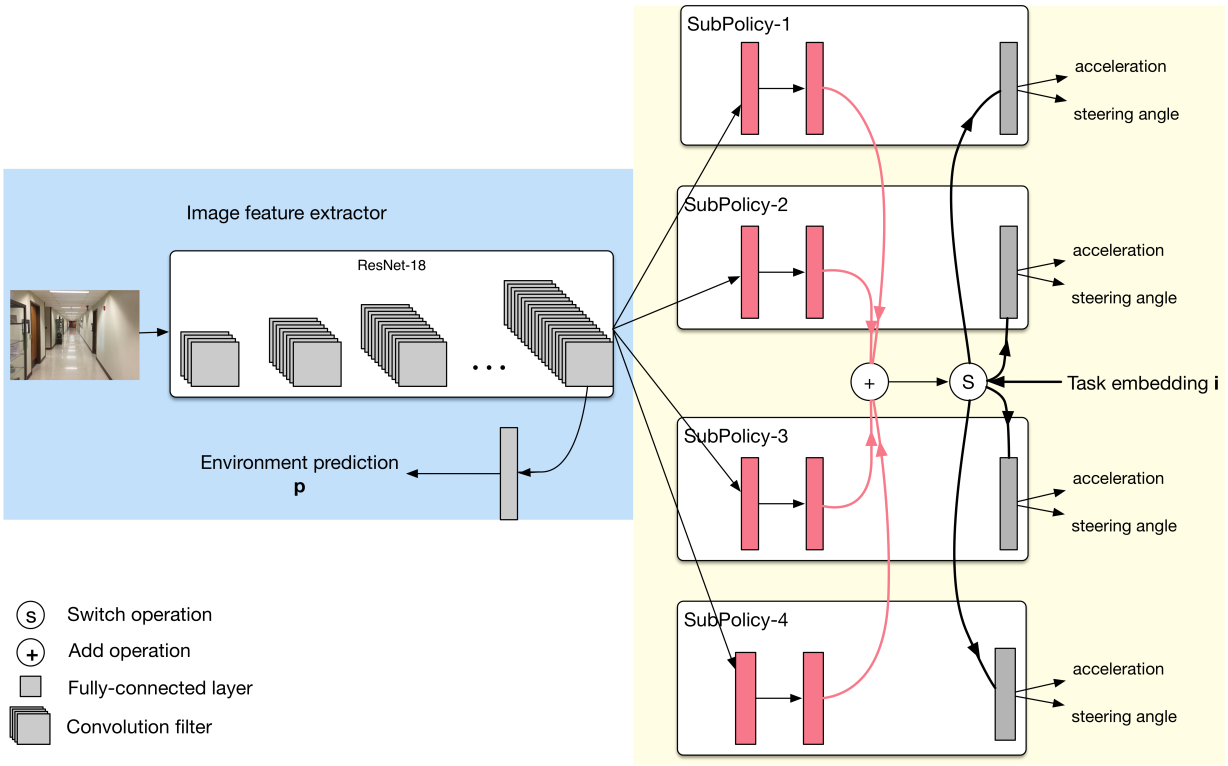



Figure 5: An illustration of SMIL framework. An image observation is first passed through a ResNet-18 to extract features. The features are then passed through a linear classifier as well as shared multi-headed policy to predict environment label and control commands.

extensively researched by the literature and it is proven to have faster convergence than training from scratch [11]. The feature vector is then flattened and passed to shared multi-headed policy to generate control commands. In addition, to allow image feature extractor to learn a more general representation of indoor environments, it also predicts which indoor environment it is currently in: $p = E(f)$, where $E$ represents a linear classifier. In our experiment, it predicts two class labels: hallway and classroom. Environment prediction is jointly trained with control commands using softmax loss function.

### 4.2.2 Shared Multi-headed Policy

Shared multi-headed policy learns shared knowledge across tasks as well as task specific knowledge. In our experiments, we define four tasks to be learned: traverse hallway, traverse classroom, to hallway, and to classroom. We denote the shared multi-headed policy as $a = \pi(f, t; \theta_\pi)$, where $\theta_\pi = \{\mathcal{W}^1, ..., \mathcal{W}^4\}$ is the parameters of the entire policy and $\mathcal{W}^i$ is the parameter set of the $i$th sub-policy. It takes two inputs: extracted features $f$ from the previous module and task embedding $t$ and outputs the control commands $a$ corresponding to the specific task. In our experiments, since we have four tasks, we denote the task embedding $t$ as a 4-dimensional one-hot vector. It is used for selecting which sub-policy to be activated. Note that $t$ can also be learned in an unsupervised way [8]. The action space is two-dimensional: acceleration and steering angle. The framework consists of three parts: switch operation, add operation, and sub-policies represented by fully connected layers.

We represent sub-policies as three-layer fully connected neural networks. Because the tasks are highly correlated, it is useful to learn task relationships through which the activated sub-policy exploits useful information from other sub-policies. For example, hallway navigation sub-policy can leverage obstacle avoidance knowledge learned by classroom navigation sub-policy because classroom is a more complicated environment with different types of obstacles. Formally, we define $\mathcal{W}_l^i$ the set of parameters at layer $l$ in $i$th sub-policy and $h_l^i = g_l(\mathcal{W}_l^{i^T} x_l^i)$ denotes the corresponding output, where function $g_l(\cdot)$ denotes non-linearity function and $x_l^i$ is the input of that layer. In our network, we set the first two layers' non-linear function as ReLU and the last layer as identity function. The input to first layers is the extracted features $f$ from the previous module.

We share the information across sub-policies using an add operation that combines all the outputs from the second layers of each sub-policy:

$$j = \sum_{i=1}^{i=T} h_2^i, \tag{5}$$

where $T$ is the total number of tasks. Although the literature has shown that higher layers learn more task specific features and they are difficult to transfer [39] , we choose to share the information from second layers because the tasks are highly correlated thus the learned features are easier to transfer over sub-policies. After the add operation, task embedding $t$ selects which sub-policy to use via a switch operation. The switch operation routes the output from the add operation to the final layer of the selected sub-policy. The final output will be

$$a = h_3^t = g_3^t(W_3^t j). \tag{6}$$

This forces each sub-policy learns the task specific controls(final layer) as well as sharing knowledge(add operation) across different sub-policies.

## 4.3 Training Procedure

This section presents the detailed training procedure for SMIL. We apply three different training techniques to train a robust SMIL framework: dropout, data augmentation, and noise injection.

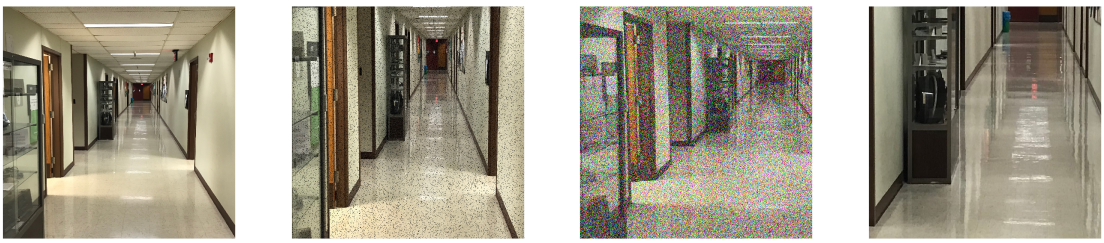


Figure 6: An illustration of randomly chosen augmentation strategies. From left to right are original, random cropping and pixel dropout, additive Gaussian noise and contrast changing, and more aggressive random cropping.

### 4.3.1 Dropout and Data Augmentation

As opposed to [38], we want to train a robust SMIL framework that is able to perform in different environments from real-world experience instead of learning from images generated by a hand-engineered simulator. It is necessary to collect images from diverse indoor environments to prevent a deep model from overfitting, but collecting data is time consuming. Hence, we apply data augmentation and dropout [33] and we found that these two strategies are crucial for training a robust model. For data augmentation, we randomly apply contrast change, Gaussian noise, pixel dropout, random cropping, and horizontal flip (steering angle is also flipped). Different augmentation strategies are shown in Fig. 6. Dropout is used to prevent model overfitting by randomly zeroing-out an neuron's activation. In addition, dropout can also stabilize performance of the robot. Because of the add operation, the norm of activation input to the final layer may becomes very large and causes the robot outputs very different control commands. Dropout is only added to the first and second layers of sub-policies.

### 4.3.2 Noise Injection

Distribution mismatch is a central problem in imitation learning: because human supervisor is proficient in demonstrating tasks, there are no demonstrations of recovering from dangerous or erroneous states. Thus, the robot does not know how to correct itself from these states. We explore off-policy training for SMIL framework to overcome this difficulty, where the robot learns from others' experience. We adopted the approach from [15], where the authors proposed a method for injecting an optimized Gaussian noise to expert policy to maximize the probability of human demonstrators making the same mistakes as the robot.

Given a robot policy $\pi_\theta$, an expert policy $\pi^*$, DART algorithm aims to find a covariance matrix of Gaussian noise that maximizes the probability of expert taking robot policy:

$$\Sigma^* = \underset{\Sigma}{arg\,min}\ E_{p(\xi|\pi^*,\Sigma)} - \sum_{t=0}^{T-1} log[\pi * (\pi_\theta|x_t, \Sigma)], \tag{7}$$

where $\xi$ means trajectories encountered by executing expert policy with noise injected and the covariance matrix is $\Sigma$. They observe that the error made by noise injected expert policy should be the same with the final robot policy. They also utilized shrinkage estimation to scale the covariance matrix and derived a closed form solution:

$$\Sigma^\alpha = \frac{\alpha}{Ttr(\Sigma^*)}\Sigma^*,\ \Sigma^* = \frac{1}{T}E_{p(\xi|\pi^*,\Sigma)}\sum_{t=0}^{T-1}(\pi_\theta(x_t) - \pi^*(x_t))(\pi_\theta(x_t) - \pi^*(x_t))^T, \tag{8}$$

where $\alpha$ is the prior knowledge of robot policy final error on the training dataset. This algorithm is best used in an iterative approach, that is we collect expert demonstrations for $k$ iterations and update the covariance matrix at the start of every iteration except

for the first iteration, which the covariance matrix will be set to 0. In our experiment, we found $\alpha = 2$ gives the best results. It is reasonable since we normalize value of steering angle and acceleration between -1 and 1, the largest $MSE$ should not exceed 4.

# 5 EXPERIMENTS

## 5.1 MS3L Performance Evaluation

The experiments have been performed in the 3rd floor of the Robert Bell (RB) Hall building at Ball State University. Fig. 7a and 7b are indoor environments where our robot has been trained and Fig.8 shows the floor plan.

(a) Hallway

(b) Classroom

Figure 7: Training environments for robots.

We have implemented *MS3L* on tensorflow [1]. Adam optimizer [13] is used and configured with a learning rate of 0.0001 for $\pi_{\theta_n}$ and 0.001 for $\pi_{\theta_r}$. Both networks are trained over 50 epochs with a $L2$ weight decay of 0.0001. We set $\gamma = 0.8$ and $\tau = 0.00025$ while training $\pi_{\theta_r}$. In addition, we set $\beta = 0.99$ in data collection using the recording policy. Input images are rescaled to $128 \times 128$ RGB images. We set $k = 4$ during self-supervised training. For each training iteration, we run the robot for $250s$ and the images are taken at $30fps$. We use the traveled *distance* and *time* to collision as evaluation

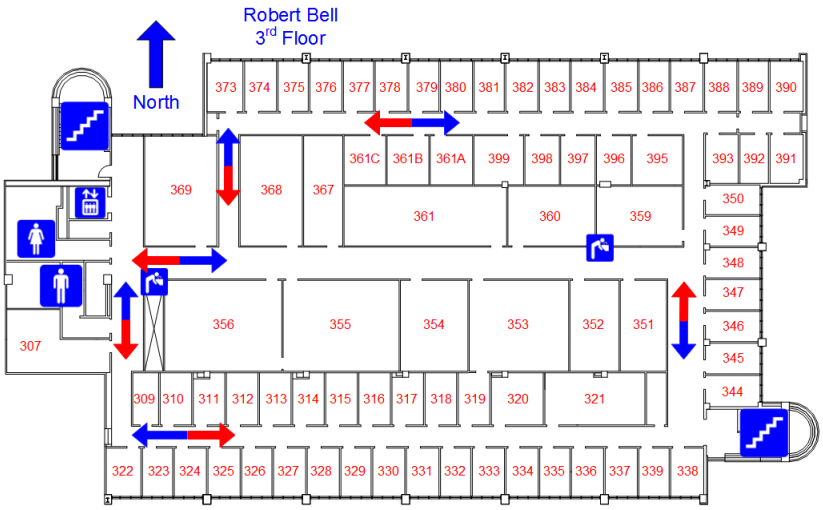


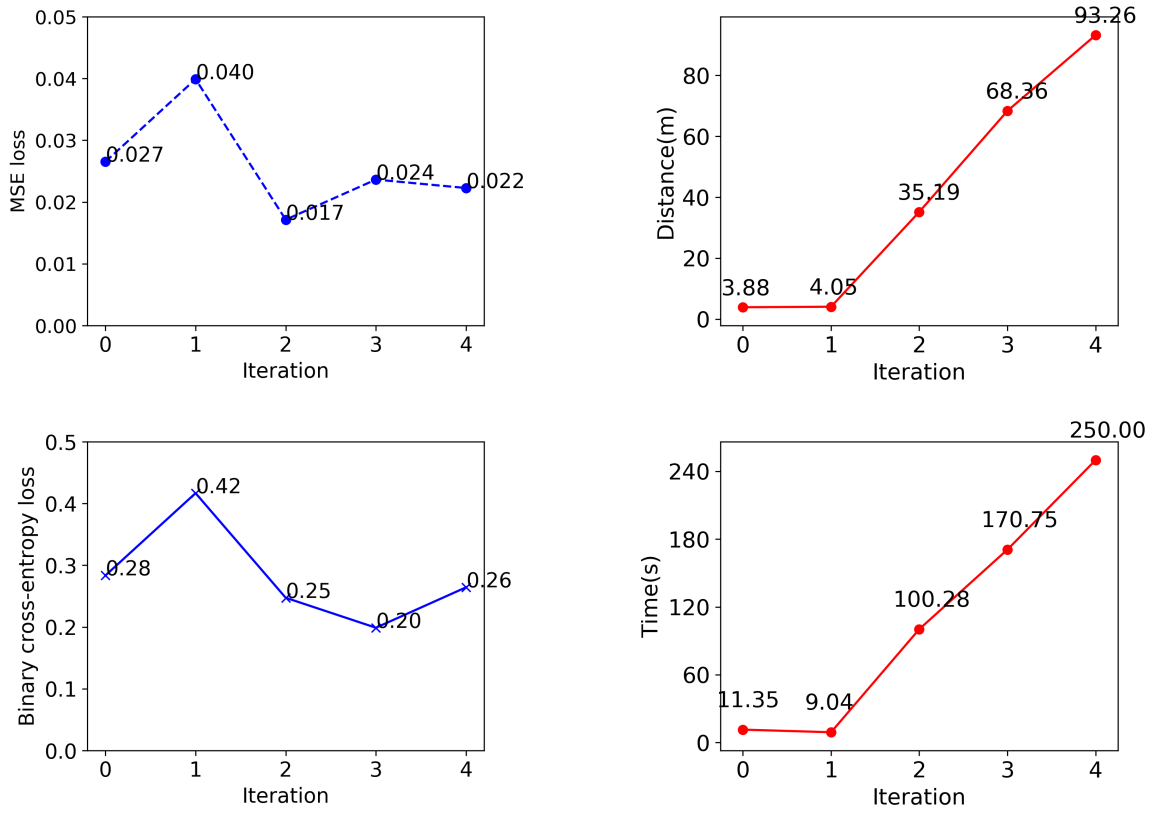

Figure 8: Data collection trajectories. Blue arrow indicates the trajectory for collecting recording policy dataset and the red one shows the trajectory for collecting navigation policy datasets for all iterations.

metrics in the experiments. We set the maximum travel duration to be $250s$ throughout the experiments.

We have trained our robot in two indoor environments: RB 3rd floor hallway as shown as red arrow in Fig. 8 and in RB-356 classroom during break time. The classroom is an extremely difficult environment, where the robot collides into obstacles even with human operator's supervision in some tests.

### 5.1.1 Training Evaluation

We first evaluate the performance of our navigation and recording policies during training process. We have reserved 10% of data samples as the validation set in each iteration. We have measured the losses for navigation and recording policies, which are the mean squared error (MSE) between reference and navigation policies and the binary cross-entropy loss respectively. They are averaged over each training iteration as shown in Fig.9a. The averaged losses of navigation and recording policies both reach their largest at




(a) (b)

Figure 9: Averaged evaluation losses on navigation **(top)** and recording policies **(bottom)** over five training iterations are shown in **(a)**. Traveled distance (**top**) and time (**bottom**) to collision in each iteration are shown in **(b)**.

the first iteration in the self-supervised stage. We conjecture this is due to the discrepancy between human and sensor policies. After this iteration, the navigation policy rapidly converges to reference policies and then fluctuates within a small range.

**Performance of Navigation Policy** Fig.9b shows the performance of navigation policy during the five training iterations. It reflects the results of averaged losses in Fig.9a: there is no performance gain in the first two iterations and then the performance linearly increases after the first iteration of the self-supervised learning phase. We have noticed that the navigation policy is able to safely navigate the robot across the hallway without any collision within the time limits at the last iteration, which is also clearly indicated

by the rightmost point on Fig.9b. These observations imply that *MS3L* can safely and autonomously navigate robots in complex real environments after sufficient iterations of training.

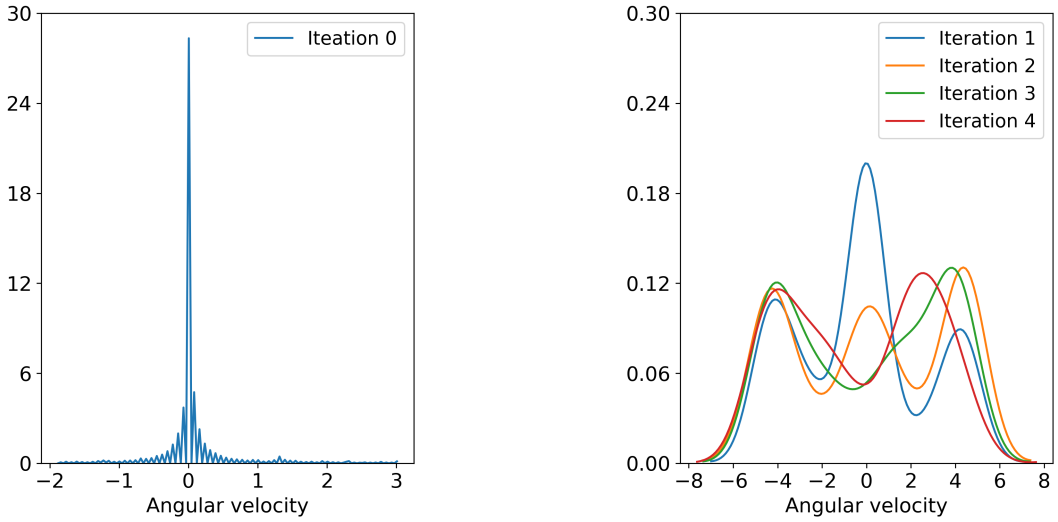
**Performance of Recording Policy** We also analyze the performance of recording policy during each iteration. As an illustration, Table 1 shows the number of training samples collected during each iteration. In the pre-training stage (Iteration 0), human operator navigates the robot to collect the initial dataset within $250s$ at $30fps$. In the self-supervised learning stage, recording policy is extremely effective in that it drastically reduces the number of collection samples by ruling out those deficient labeled observations.

Table 1: Training data collected during 5 iterations

| Iteration | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| # of observations | 7500 | 351 | 853 | 790 | 335 |

In addition, distributions for angular velocity during each iteration are are another metric to measure the performance of the recording policy. They are evaluated and presented in Fig.10. The training samples collected in the pre-training stage primarily consist of data with angular velocity $0rad/s$ as in Fig.10a, which represents going forward. This singular value can't contribute anything useful to data labeling for training. In contrast, the data collected is more uniformly distributed in self-supervised learning stage as in Fig.10b. This means these data contain more stateful information such as turning in different angular velocities and are thus significantly useful for training. This also indicates that, in addition to reducing the effect of deficient labeling from sensor policy, *MS3L* also ensures only hard samples that have not been seen in the past iterations are

recorded and trained by navigation policy.




(a) (b)

Figure 10: Angular velocity distributions over the pre-training stage (**a**) and self-supervised learning stage (**b**).

### 5.1.2 Recording Threshold $\beta$

An important factor that affects the framework and recording policy is threshold $\beta$ that controls what observations should be considered as hard samples. We use three different $\beta$ values to evaluate how it influences the performance of the system as shown in Fig.11. The performance has been evaluated at the third iteration of training process[4]. As we can observe from the figure, the traveled distance decreases as we reduce $\beta$. We conjecture that this is because the system learns too much from the suboptimal sensor policy with a small $\beta$. When $\beta$ is near 0, the recording policy is merely utilized and the data collection is not constrained, which degrades the performance of navigation policy. A large $\beta$ value is therefore necessary to ensure the recording policy to perform effectively during the self-supervised learning stage.

[4]Evaluating the performance after the third iteration is dangerous when $\beta$ is 0.5 or 0.1 because the performance of navigation policy degrades quickly.

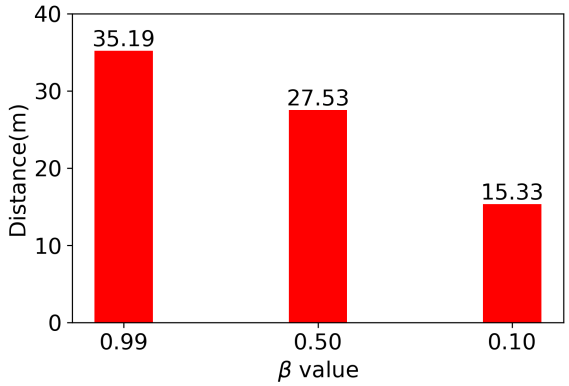



Figure 11: Traveled distances with $\beta$ values of 0.99, 0.5, and 0.1.

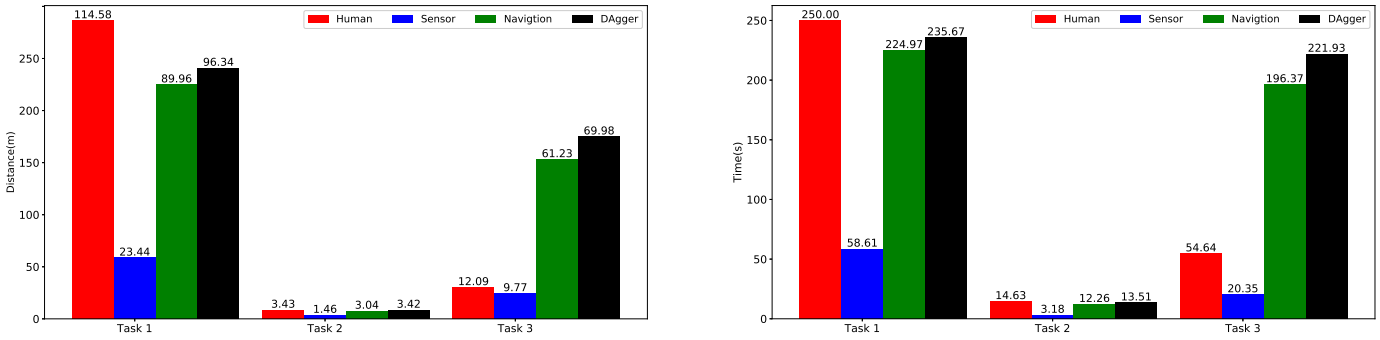
### 5.1.3 Performance of Navigation After Training: Comparison with Baselines

After training, it is of utmost interest to assess the actual performance of the trained *MS3L* robot in real environments. Since the robot has been trained and learned from human and sensor policies, we define these two policies as baselines. In addition, DAgger algorithm is also used to compare with our framework. The training process of DAgger is the same as our framework except that it requires all 37500 images to be recorded during the five training iterations and it needs human to correct the actions. It is valuable to compare the robot navigation performance with these baselines. We have tested the comparison over three tasks. For fair comparison, during the tests, the human operator can only perceive the environments at the first-person view from the cameras of the robot as it does at its own. The results show that *MS3L* is robust to learn from noisy and suboptimal policy. We have also noticed that *MS3L* surpasses the sensor policy by a large margin in most of the tasks, it is also able to achieve near-human performance in two tasks, and even outperforms the human operator in one task. Two human operators participate in each task. Every task is performed 5 times and the results are shown in average distance in meters and time in seconds. The three tasks are presented as follows:

- The first task is to navigate the robot through the hallway during normal business time. While our robot is trained in this environment, the training data collected is during break time when there are few people walking in the environment. We set this task during business hours when the walking traffic is heavy inside the hallways to examine how robust *MS3L* is to deal with unseen cases after training.

- The second task is navigate the robot through a classroom where chairs and tables are main obstacles. This environment is difficult even for human operator.

- The third task is navigate the robot through the 3rd floor with Gaussian noise with zero mean and unit standard deviation added to the controller that emulates hardware malfunction. We would like to use this test to show the robustness of *MS3L* in case of hardware failures or human manipulation mistakes.




(a)

(b)

Figure 12: Traveled distance (**a**) and time to collision (**b**) in the test stage after training in three tasks.

Fig.12 shows the results of comparisons in these three tasks. We can observe that the performance of *MS3L* exceeds sensor policy in every environment tested. Human policy as the baseline has a slightly higher performance in the first tasks. However, in case of

hardware failures or operation mistakes as in the third task, human operator is not able to deal with such unexpected events, while *MS3L* is robust to these noises and greatly surpasses human performance. DAgger achieves slightly better performances in all three tasks compared to *MS3L*. However, DAgger requires 2.8× more examples to be recorded compared to our framework. In addition, it needs heavy human labeling which does not exist in our work.

From the experiments, the sensor policy often fails when the robot faces a plain wall or narrow corridor where depth information can not be reliably estimated from stereo images correctly. The navigation policy fails in avoiding multiple objects. This is because the environment is not completely observable and the policy only considers the current observation, not any historical information. For example, it forgets the position of the previous object while trying to avoid the current one.

## 5.2 $SMIL$ Performance Evaluation

We evaluate $SMIL$ framework in indoor environments in Robert Bell building. Our experiments are designed to answer the following:

1. Is learning shared task representation (add operation) necessary for multi-task imitation learning when the tasks are highly correlated?

2. How does the multi-headed sub-policy architecture compare to a single-headed policy?

3. Does adding environment prediction task improve performance?

4. Are data augmentation and dropout important to generalization and robustness?

### 5.2.1 Task Description

We evaluate our framework on four correlated tasks in Robert Bell building and are described in Table 2. Since classroom indoor environments contain fewer free space and are far more complex compared to hallway environments, we set less constraints on classroom related task, i.e. traverse classroom and to hallway. The goal of these four tasks is to simulate the robot in a multi-task decision making environment, where it is required to go to different indoor locations based on human command.

Table 2: Task Description

| Task | Task Description | Time Limit | Failure Condition |
|---|---|---|---|
| Traverse Hallway | The robot is initialized in hallway at a fixed position. It is asked to traverse hallway without collision within the time limit. | 1 min | If the robot collides into obstacles, we count it as a failure. If it goes into classroom, we count it as a failure. |
| Traverse classroom | The robot is initialized in classroom at the door position. It is asked to traverse classroom without collision within the time limit | 30 sec | Because classroom is highly complex, we give the robot one more chance in this task. If the robot collides into obstacles, we reroute it back to free space. If the robot collides again, we count it as a failure. In addition, if it goes outside of the classroom, we count it as a failure. |
| To classroom | The robot is initialized in hallway at a fixed position that is 15 meters away from a classroom. It needs to go from the classroom to the hallway. | 2min | If the robot can not complete the task within time limit or collide into obstacles, we count it as a failure. In addition, if the robot passes two nearest classrooms from its initial position, it is a failure. |
| To hallway | The robot is initialized in a classroom at the furthest corner away from the door. It needs to go through the classroom to the hallway. | 1 min | If the robot can not complete the task within time limit or collide into obstacles, we count it as a failure. Same as traverse classroom, we give the robot a second chance. |

### 5.2.2 Experiment Setup

In our experiments, we use the same robot as in Section 3.1, but only RGB images from the stereo camera are used. A PlayStation controller is used to send control and task commands to the robot.

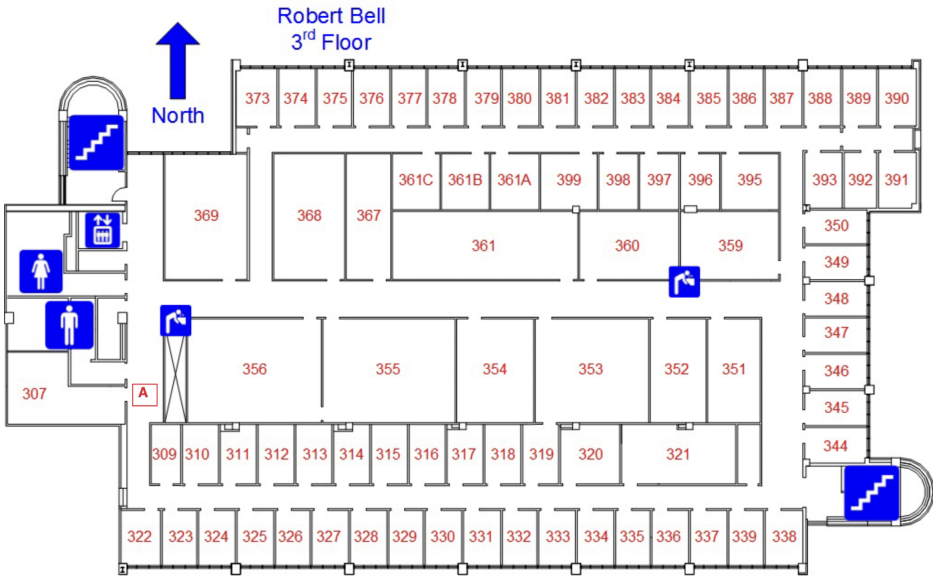



Figure 13: Data collection trajectories. Blue arrow indicates the trajectory for collecting recording policy dataset and the red one shows the trajectory for collecting navigation policy datasets for all iterations.

We collect our training data on the third floor of Robert Bell building. It consists of 10 minutes driving for each task, which corresponds to roughly 20,000 images each task. We use four DART iterations to collect our training dataset per task, so one DART iteration corresponds to 2.5 minutes driving (4500 images) for each task. In the experiments, we found it is hard for humans to control when apply the Gaussian noise to all actions. Thus, for each image, there is 50% probability to add the optimized Gaussian noise. For noise injected images, we only record the corrective actions taken by human demonstrator not the noise itself. Images are recorded at the resolution of $224 \times 224$ to be consistent with the original ResNet-18 setup. When switching to different tasks, human demonstrator presses physical buttons on PlayStation controller to provide a task index and a script is used to map this index to a one-hot task embedding vector.

For traverse hallway task, we place our robot at position $A$ and drive it clockwise at every DART iteration. Same strategy is applied to to classroom task. At every iteration,

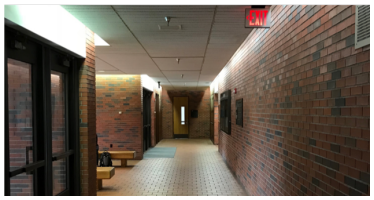
the robot starts at position $A$and runs clockwise. The human demonstrator drives the robot into the first open room that the robot encounters. For traverse classroom task, we collect data in classroom 307, 356, 354, and 360 at every DART iteration. The data of to hallway task is also recorded in the same classroom. The human demonstrator drives the robot from the most inner corner of each classroom to the hallway.

### 5.2.3 Results

We set up a variety of baselines to answer the questions asked at the beginning of this section. We compare $SMIL$ full architecture with five baselines:

- **Multi-headed network**: this model uses the same architecture without learning shared representation (excludes the add operation).
- **Plain network**: this is the traditional imitation learning approach, where there is one single network that maps input to output control commands. It is not aware of the multi-task setting.
- $SMIL$ **w/o data augmentation**: this baseline excludes data augmentation while training.
- $SMIL$ **w/o dropout**: this baseline removes dropout at the first and second layers of sub-policies.
- $SMIL$ **w/o environment prediction**: this model does not predict environment class labels.

We use the same hyper-parameters to train each baseline network (will be introduced in Section 5.2.3) across all the tasks. We use SGD optimizer with learning rate 0.01 and



(a) Robert Bell first floor hallway.

(b) Robert Bell third floor hallway.

(c) Robert Bell first floor classroom.

(d) Robert Bell third floor classroom.

Figure 14: Training environments (b) and (d) have different geometric and color appearances as testing environments (a) and (c).

decay a factor of 10 for every 5 epochs. We train each network for 30 epochs and a batch size of 256. For $SMIL$ framework and multi-headed network, we split training examples of each task evenly in one batch. We train the plain network on all the training data. The plain network is task-agnostic, so we do not tell it which task to perform explicitly. We set weight decay equals to 0.0005. For the networks that use dropout, we set dropout rate as 0.2 meaning there is a 20% probability that a neuron's activation will be set to 0.

Table 3: Success Rate

| | Traverse Hallway | Traverse Classroom | To Classroom | To Hallway |
|---|---|---|---|---|
| ***SMIL*** | **80%** | **60%** | **70%** | **60%** |
| Multi-headed | 50% | 30% | 50% | 40% |
| Plain | 50% | 30% | 20% | 0% |
| $SMIL$ w/o augmentation | 40% | 30% | 40% | 30% |
| $SMIL$ w/o dropout | 30% | 30% | 50% | 40% |
| $SMIL$ w/o environment | 70% | 50% | 50% | 50% |

To test generalization, we run the robot on the first floor of Robert Bell building at test time. The geometric and color appearances are not the same as the training

environments. The testing and training scenes are illustrated in Fig. 14. We conducted 10 experiments for each task and recorded success rate and averaged time duration with standard deviation for each experiment as shown in Table 3 and Fig. 15.

**Overall Comparisons:** In terms of success rate, it is obvious that our proposed model, $SMIL$, achieves the best performance across all the tasks. In terms of time duration, our model is able to maintain the longest averaged travels among all other models in traverse tasks. It needs slightly more time to complete to classroom task. This is because some baseline models complete all easy runs that take less time, e.g. the classrooms that are in the robot's field of view, but failed difficult runs that take longer time.

**Comparisons on model architectures:** The comparisons address the first three questions at the beginning of this section. **1**. The first floor has more obstacles than the third floor, which is used for training. Thus, it is necessary to reuse the knowledge learned from traverse classroom task while performing traverse hallway task. By comparing $SMIL$ and multi-headed network on traverse hallway task, we observe that $SMIL$ is able to reuse obstacle avoidance knowledge from traverse classroom sub-policy. **2**. From the result, we observe that it is necessary to add this environment prediction auxiliary task to provide a denser training single to image feature extractor that allows the image feature extractor to learn a more robust representation. **3**. Although the plain network is trained on all the data which contains 80,000 images, it still failed tremendously especially on the to classroom/hallway task. This is because of mode averaging [2] and inaccurate labeling that causes bias in the network. To classroom/hallway tasks inherent a different mode than traverse classroom/hallway tasks and they are more difficult. Since the plain

network is trained on all tasks, it is possible that the plain network ignores differences across each task and thus yield an undesirable policy. In addition, it is task-agnostic, so it can not respond to human command.

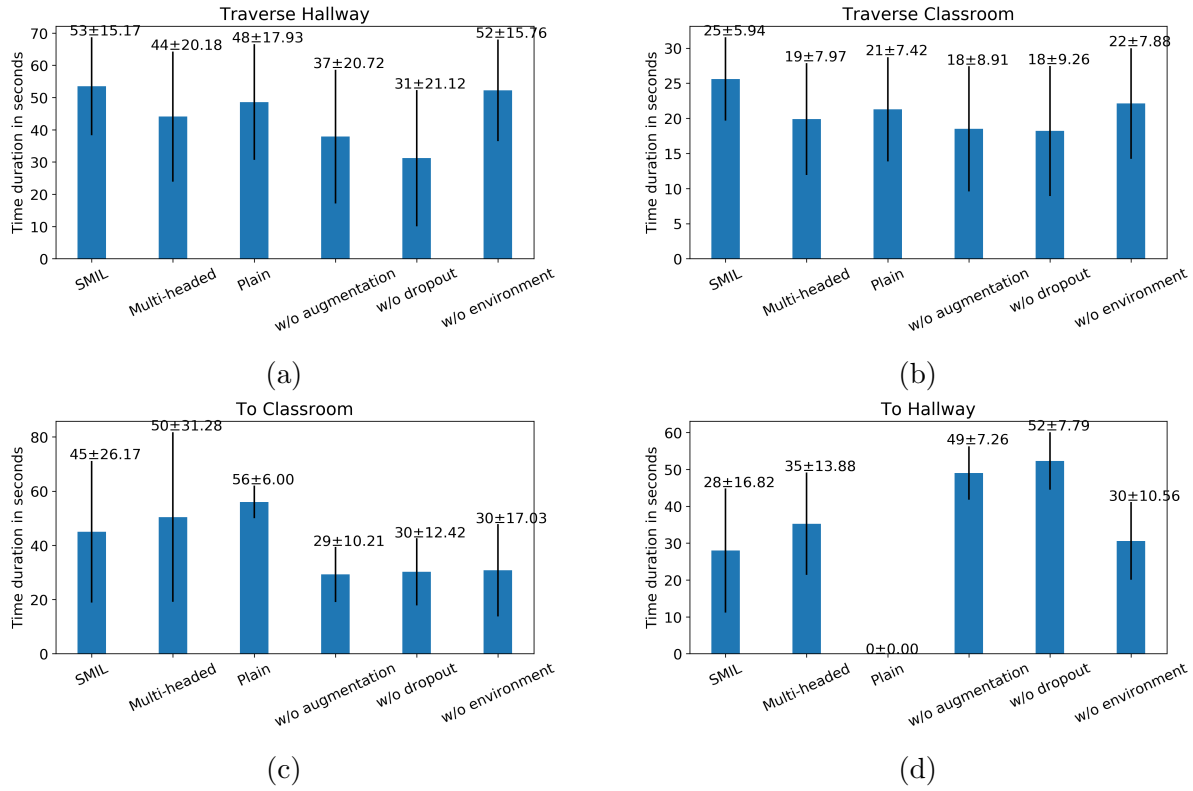



Figure 15: Averaged time duration and standard deviation for each task. For traverse hallway/classroom task, The higher the averaged time duration means the the robot gets less collision. For to hallway/classroom tasks, the lower the averaged time duration means the faster the robot completes the task.

**Comparison on generalization and robustness:** This comparison answers the fourth question: both dropout and data augmentation are necessary to train a robust model. By removing a single method, the robot has the similar performance. In terms of success rate, after adding both methods, the robot's performance is almost doubled. It shows that these two methods are complementary to each other. Dropout prevents the robot from aggressive turning caused by large activation from add operation. By augmenting

the training dataset, the trained model learns to ignore geometric difference and different lighting effects.

# 6 CONCLUSION

In this thesis, we proposed two novel frameworks that give the robot the ability to navigate and perform different tasks in indoor environments. The first framework is Multi-Sensory Self- Supervised Learning(*MS3L*). *MS3L* combines deep imitation learning with sensor fusion. It learns from human demonstrations to perform robotic navigation without a predefined map. *MS3L* designs a suboptimal sensor policy that replaces human operators after the initial training. A recording policy is then proposed to restrict learning from the suboptimal policy that likely lead to serious robotic damage. The second framework, *SMIL*, addresses multi-task imitation learning problem. Our framework learns task relationships by summing up all activations output from the second layer of each sub-policy. At training time, dropout and data augmentation are applied to train a robust model that generalizes well in unseen environments. At test time, a task embedding is used to activate one sub-policy based on human command. By leveraging shared knowledge, our framework is able to perform better than multi-headed framework. Extensive experiments in real indoor environments have demonstrated that both *MS3L* and *SMIL* are able to successfully and reliably apply to real indoor environments.

# REFERENCES


[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] C. M. Bishop. Mixture density networks. 1994.

[3] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[4] R. Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998.

[5] F. Codevilla, M. Müller, A. Dosovitskiy, A. López, and V. Koltun. End-to-end driving via conditional imitation learning. *arXiv preprint arXiv:1710.02410*, 2017.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[7] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive Mapping and Planning for Visual Navigation. 2017.

[8] K. Hausman, Y. Chebotar, S. Schaal, G. Sukhatme, and J. J. Lim. Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 1235–1245, 2017.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[11] M. Huh, P. Agrawal, and A. A. Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[12] D. Isele, A. Cosgun, K. Subramanian, and K. Fujimura. Navigating Intersections with Autonomous Vehicles using Deep Reinforcement Learning. *arXiv preprint arXiv:1705.01196*, 2017.

[13] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *Computer Science*, 2014.

[14] M. Laskey, J. Lee, C. Chuck, D. Gealy, W. Hsieh, F. T. Pokorny, A. D. Dragan, and K. Goldberg. Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations. In *IEEE International Conference on Automation Science and Engineering*, pages 827–834, 2016.

[15] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on Robot Learning*, pages 143–156, 2017.

[16] M. Laskey, S. Staszak, W. Y.-S. Hsieh, J. Mahler, F. T. Pokorny, A. D. Dragan, and K. Goldberg. Shiv: Reducing supervisor burden in dagger using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *Robotics*

*and Automation (ICRA), 2016 IEEE International Conference on*, pages 462–469. IEEE, 2016.

[17] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[18] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *Computer Science*, 8(6):A187, 2015.

[20] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

[21] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[23] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena. From Perception to Decision: A Data-driven Approach to End-to-end Motion Planning for Autonomous Ground Robots. *arXiv preprint arXiv:1609.07910*, 2016.

[24] R. Rahmatizadeh, P. Abolghasemi, A. Behal, and L. Bölöni. Learning real manipulation tasks from virtual demonstrations using LSTM. *arXiv preprint arXiv:1603.03833*, 2016.

[25] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on robotics and automation*, 8(5):501–518, 1992.

[26] S. Ross, G. J. Gordon, and D. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *AISTATS*, volume 1, page 6, 2011.

[27] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert. Learning monocular reactive uav control in cluttered natural environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1765–1772. IEEE, 2013.

[28] S. Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[30] F. Sadeghi and S. Levine. $(CAD)^2$ RL: Real Single-Image Flight without a Single Real Image. *arXiv preprint arXiv:1611.04201*, 2016.

[31] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[33] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[34] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[35] L. Tai, G. Paolo, and M. Liu. Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation. *arXiv preprint arXiv:1703.00420*, 2017.

[36] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts. LQR-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010.

[37] S. Thrun and J. J. Leonard. Simultaneous localization and mapping. In *Springer handbook of robotics*, pages 871–889. Springer, 2008.

[38] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 23–30. IEEE, 2017.

[39] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[40] J. Zhang and K. Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016.

[41] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. 2016.